NAME

Context Management −

Functions

CUresult cuCtxCreate (**CUcontext** *pctx, unsigned int flags, **CUdevice** dev)
Create a CUDA context.

CUresult cuCtxDestroy (**CUcontext** ctx)
Destroy a CUDA context.

CUresult cuCtxGetApiVersion (**CUcontext** ctx, unsigned int *version)
Gets the context's API version.

CUresult cuCtxGetCacheConfig (**CUfunc_cache** *pconfig)
Returns the preferred cache configuration for the current context.

CUresult cuCtxGetCurrent (**CUcontext** *pctx)
Returns the CUDA context bound to the calling CPU thread.

CUresult cuCtxGetDevice (**CUdevice** *device)
Returns the device ID for the current context.

CUresult cuCtxGetLimit (size_t *pvalue, **CUlimit** limit)
Returns resource limits.

CUresult cuCtxGetSharedMemConfig (**CUsharedconfig** *pConfig)
Returns the current shared memory configuration for the current context.

CUresult cuCtxGetStreamPriorityRange (int *leastPriority, int *greatestPriority)
Returns numerical values that correspond to the least and greatest stream priorities.

CUresult cuCtxPopCurrent (**CUcontext** *pctx)
Pops the current CUDA context from the current CPU thread.

CUresult cuCtxPushCurrent (**CUcontext** ctx)
Pushes a context on the current CPU thread.

CUresult cuCtxSetCacheConfig (**CUfunc_cache** config)
Sets the preferred cache configuration for the current context.

CUresult cuCtxSetCurrent (**CUcontext** ctx)
Binds the specified CUDA context to the calling CPU thread.

CUresult cuCtxSetLimit (**CUlimit** limit, size_t value)
Set resource limits.

CUresult cuCtxSetSharedMemConfig (**CUsharedconfig** config)
Sets the shared memory configuration for the current context.

CUresult cuCtxSynchronize (void)
Block for a context's tasks to complete.

Detailed Description

\brief context management functions of the low-level CUDA driver API (**cuda.h**)

This section describes the context management functions of the low-level CUDA driver application programming interface.

Function Documentation

CUresult cuCtxCreate (CUcontext * pctx, unsigned int flags, CUdevice dev)

Creates a new CUDA context and associates it with the calling thread. The `flags` parameter is described below. The context is created with a usage count of 1 and the caller of **cuCtxCreate()** must call **cuCtxDestroy()** or when done using the context. If a context is already current to the thread, it is supplanted by the newly created context and may be restored by a subsequent call to **cuCtxPopCurrent()**.

The three LSBs of the `flags` parameter can be used to control how the OS thread, which owns the CUDA context at the time of an API call, interacts with the OS scheduler when waiting for results from the GPU. Only one of the scheduling flags can be set when creating a context.

- **CU_CTX_SCHED_AUTO**: The default value if the `flags` parameter is zero, uses a heuristic based on the number of active CUDA contexts in the process *C* and the number of logical processors in the system *P*. If *C* > *P*, then CUDA will yield to other OS threads when waiting for the GPU, otherwise CUDA will not yield while waiting for results and actively spin on the processor.
- **CU_CTX_SCHED_SPIN**: Instruct CUDA to actively spin when waiting for results from the GPU. This can decrease latency when waiting for the GPU, but may lower the performance of CPU threads if they are performing work in parallel with the CUDA thread.

- **CU_CTX_SCHED_YIELD**: Instruct CUDA to yield its thread when waiting for results from the GPU. This can increase latency when waiting for the GPU, but can increase the performance of CPU threads performing work in parallel with the GPU.
- **CU_CTX_SCHED_BLOCKING_SYNC**: Instruct CUDA to block the CPU thread on a synchronization primitive when waiting for the GPU to finish work.
- **CU_CTX_BLOCKING_SYNC**: Instruct CUDA to block the CPU thread on a synchronization primitive when waiting for the GPU to finish work.
  Deprecated: This flag was deprecated as of CUDA 4.0 and was replaced with **CU_CTX_SCHED_BLOCKING_SYNC**.
- **CU_CTX_MAP_HOST**: Instruct CUDA to support mapped pinned allocations. This flag must be set in order to allocate pinned host memory that is accessible to the GPU.
- **CU_CTX_LMEM_RESIZE_TO_MAX**: Instruct CUDA to not reduce local memory after resizing local memory for a kernel. This can prevent thrashing by local memory allocations when launching many kernels with high local memory usage at the cost of potentially increased memory usage.

Context creation will fail with **CUDA_ERROR_UNKNOWN** if the compute mode of the device is **CU_COMPUTEMODE_PROHIBITED**. Similarly, context creation will also fail with **CUDA_ERROR_UNKNOWN** if the compute mode for the device is set to **CU_COMPUTEMODE_EXCLUSIVE** and there is already an active context on the device. The function **cuDeviceGetAttribute()** can be used with **CU_DEVICE_ATTRIBUTE_COMPUTE_MODE** to determine the compute mode of the device. The *nvidia-smi* tool can be used to set the compute mode for devices. Documentation for *nvidia-smi* can be obtained by passing a −h option to it.

Parameters:
pctx - Returned context handle of the new context
flags - Context creation flags
dev - Device to create context on

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_DEVICE**, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_OUT_OF_MEMORY**, **CUDA_ERROR_UNKNOWN**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxDestroy, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**, **cuCtxSynchronize**

CUresult cuCtxDestroy (CUcontext ctx)

Destroys the CUDA context specified by `ctx`. The context `ctx` will be destroyed regardless of how many threads it is current to. It is the responsibility of the calling function to ensure that no API call issues using `ctx` while **cuCtxDestroy()** is executing.

If `ctx` is current to the calling thread then `ctx` will also be popped from the current thread's context stack (as though **cuCtxPopCurrent()** were called). If `ctx` is current to other threads, then `ctx` will remain current to those threads, and attempting to access `ctx` from those threads will result in the error **CUDA_ERROR_CONTEXT_IS_DESTROYED**.

Parameters:
ctx - Context to destroy

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**, **cuCtxSynchronize**

CUresult cuCtxGetApiVersion (CUcontext ctx, unsigned int * version)

Returns a version number in `version` corresponding to the capabilities of the context (e.g. 3010 or 3020), which library developers can use to direct callers to a specific API version. If `ctx` is NULL, returns the API version used to create the currently bound context.

Note that new API versions are only introduced when context capabilities are changed that break binary compatibility, so the API version and driver version may be different. For example, it is valid for the API version to be 3020 while the driver version is 4020.

Parameters:
ctx - Context to check
version - Pointer to version

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_UNKNOWN**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**, **cuCtxSynchronize**

CUresult cuCtxGetCacheConfig (CUfunc_cache * pconfig)

On devices where the L1 cache and shared memory use the same hardware resources, this function returns through `pconfig` the preferred cache configuration for the current context. This is only a preference. The driver will use the requested configuration if possible, but it is free to choose a different configuration if required to execute functions.

This will return a `pconfig` of **CU_FUNC_CACHE_PREFER_NONE** on devices where the size of the L1 cache and shared memory are fixed.

The supported cache configurations are:

- **CU_FUNC_CACHE_PREFER_NONE**: no preference for shared memory or L1 (default)
- **CU_FUNC_CACHE_PREFER_SHARED**: prefer larger shared memory and smaller L1 cache
- **CU_FUNC_CACHE_PREFER_L1**: prefer larger L1 cache and smaller shared memory
- **CU_FUNC_CACHE_PREFER_EQUAL**: prefer equal sized L1 cache and shared memory

Parameters:
pconfig - Returned cache configuration

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**, **cuCtxSynchronize**, **cuFuncSetCacheConfig**

CUresult cuCtxGetCurrent (CUcontext * pctx)

Returns in `*pctx` the CUDA context bound to the calling CPU thread. If no context is bound to the calling CPU thread then `*pctx` is set to NULL and **CUDA_SUCCESS** is returned.

Parameters:

pctx - Returned context handle

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**,

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxSetCurrent, **cuCtxCreate**, **cuCtxDestroy**

CUresult cuCtxGetDevice (CUdevice * device)
Returns in `*device` the ordinal of the current context's device.

Parameters:
device - Returned device ID for the current context

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**,

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**, **cuCtxSynchronize**

CUresult cuCtxGetLimit (size_t * pvalue, CUlimit limit)
Returns in `*pvalue` the current size of `limit`. The supported **CUlimit** values are:

- **CU_LIMIT_STACK_SIZE**: stack size in bytes of each GPU thread.
- **CU_LIMIT_PRINTF_FIFO_SIZE**: size in bytes of the FIFO used by the printf() device system call.
- **CU_LIMIT_MALLOC_HEAP_SIZE**: size in bytes of the heap used by the malloc() and free() device system calls.
- **CU_LIMIT_DEV_RUNTIME_SYNC_DEPTH**: maximum grid depth at which a thread can issue the device runtime call cudaDeviceSynchronize() to wait on child grid launches to complete.
- **CU_LIMIT_DEV_RUNTIME_PENDING_LAUNCH_COUNT**: maximum number of outstanding device runtime launches that can be made from this context.

Parameters:
limit - Limit to query
pvalue - Returned size of limit

Returns:
CUDA_SUCCESS, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_UNSUPPORTED_LIMIT**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**, **cuCtxSynchronize**

CUresult cuCtxGetSharedMemConfig (CUsharedconfig * pConfig)
This function will return in `pConfig` the current size of shared memory banks in the current context. On devices with configurable shared memory banks, **cuCtxSetSharedMemConfig** can be used to change this setting, so that all subsequent kernel launches will by default use the new bank size. When **cuCtxGetSharedMemConfig** is called on devices without configurable shared memory, it will return the fixed bank size of the hardware.

The returned bank configurations can be either:

- **CU_SHARED_MEM_CONFIG_FOUR_BYTE_BANK_SIZE**: shared memory bank width is four bytes.
- **CU_SHARED_MEM_CONFIG_EIGHT_BYTE_BANK_SIZE**: shared memory bank width will eight bytes.

Parameters:
pConfig - returned shared memory configuration

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetLimit**, **cuCtxSynchronize**, **cuCtxGetSharedMemConfig**, **cuFuncSetCacheConfig**,

CUresult cuCtxGetStreamPriorityRange (int * leastPriority, int * greatestPriority)

Returns in `*leastPriority` and `*greatestPriority` the numerical values that correspond to the least and greatest stream priorities respectively. Stream priorities follow a convention where lower numbers imply greater priorities. The range of meaningful stream priorities is given by [`*greatestPriority`, `*leastPriority`]. If the user attempts to create a stream with a priority value that is outside the meaningful range as specified by this API, the priority is automatically clamped down or up to either `*leastPriority` or `*greatestPriority` respectively. See **cuStreamCreateWithPriority** for details on creating a priority stream. A NULL may be passed in for `*leastPriority` or `*greatestPriority` if the value is not desired.

This function will return ’0’ in both `*leastPriority` and `*greatestPriority` if the current context’s device does not support stream priorities (see **cuDeviceGetAttribute**).

Parameters:
leastPriority - Pointer to an int in which the numerical value for least stream priority is returned
greatestPriority - Pointer to an int in which the numerical value for greatest stream priority is returned

Returns:
CUDA_SUCCESS, **CUDA_ERROR_INVALID_VALUE**,

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuStreamCreateWithPriority, **cuStreamGetPriority**, **cuCtxGetDevice**, **cuCtxSetLimit**, **cuCtxSynchronize**

CUresult cuCtxPopCurrent (CUcontext * pctx)

Pops the current CUDA context from the CPU thread and passes back the old context handle in `*pctx`. That context may then be made current to a different CPU thread by calling **cuCtxPushCurrent()**.

If a context was current to the CPU thread before **cuCtxCreate()** or **cuCtxPushCurrent()** was called, this function makes that context current to the CPU thread again.

Parameters:
pctx - Returned new context handle

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**, **cuCtxSynchronize**

CUresult cuCtxPushCurrent (CUcontext ctx)

Pushes the given context `ctx` onto the CPU thread's stack of current contexts. The specified context becomes the CPU thread's current context, so all CUDA functions that operate on the current context are affected.

The previous current context may be made current again by calling **cuCtxDestroy()** or **cuCtxPopCurrent()**.

Parameters:
ctx - Context to push

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**, **cuCtxSynchronize**

CUresult cuCtxSetCacheConfig (CUfunc_cache config)

On devices where the L1 cache and shared memory use the same hardware resources, this sets through `config` the preferred cache configuration for the current context. This is only a preference. The driver will use the requested configuration if possible, but it is free to choose a different configuration if required to execute the function. Any function preference set via **cuFuncSetCacheConfig()** will be preferred over this context-wide setting. Setting the context-wide cache configuration to **CU_FUNC_CACHE_PREFER_NONE** will cause subsequent kernel launches to prefer to not change the cache configuration unless required to launch the kernel.

This setting does nothing on devices where the size of the L1 cache and shared memory are fixed.

Launching a kernel with a different preference than the most recent preference setting may insert a device-side synchronization point.

The supported cache configurations are:

- **CU_FUNC_CACHE_PREFER_NONE**: no preference for shared memory or L1 (default)
- **CU_FUNC_CACHE_PREFER_SHARED**: prefer larger shared memory and smaller L1 cache
- **CU_FUNC_CACHE_PREFER_L1**: prefer larger L1 cache and smaller shared memory
- **CU_FUNC_CACHE_PREFER_EQUAL**: prefer equal sized L1 cache and shared memory

Parameters:
config - Requested cache configuration

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetLimit**, **cuCtxSynchronize**, **cuFuncSetCacheConfig**

CUresult cuCtxSetCurrent (CUcontext ctx)

Binds the specified CUDA context to the calling CPU thread. If `ctx` is NULL then the CUDA context previously bound to the calling CPU thread is unbound and **CUDA_SUCCESS** is returned.

If there exists a CUDA context stack on the calling CPU thread, this will replace the top of that stack with `ctx`. If `ctx` is NULL then this will be equivalent to popping the top of the calling CPU thread's CUDA context stack (or a no-op if the calling CPU thread's CUDA context stack is empty).

Parameters:

ctx - Context to bind to the calling CPU thread

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxGetCurrent, **cuCtxCreate**, **cuCtxDestroy**

CUresult cuCtxSetLimit (CUlimit limit, size_t value)

Setting `limit` to `value` is a request by the application to update the current limit maintained by the context. The driver is free to modify the requested value to meet h/w requirements (this could be clamping to minimum or maximum values, rounding up to nearest element size, etc). The application can use **cuCtxGetLimit()** to find out exactly what the limit has been set to.

Setting each **CUlimit** has its own specific restrictions, so each is discussed here.

- **CU_LIMIT_STACK_SIZE** controls the stack size in bytes of each GPU thread. This limit is only applicable to devices of compute capability 2.0 and higher. Attempting to set this limit on devices of compute capability less than 2.0 will result in the error **CUDA_ERROR_UNSUPPORTED_LIMIT** being returned.
- **CU_LIMIT_PRINTF_FIFO_SIZE** controls the size in bytes of the FIFO used by the printf() device system call. Setting **CU_LIMIT_PRINTF_FIFO_SIZE** must be performed before launching any kernel that uses the printf() device system call, otherwise **CUDA_ERROR_INVALID_VALUE** will be returned. This limit is only applicable to devices of compute capability 2.0 and higher. Attempting to set this limit on devices of compute capability less than 2.0 will result in the error **CUDA_ERROR_UNSUPPORTED_LIMIT** being returned.
- **CU_LIMIT_MALLOC_HEAP_SIZE** controls the size in bytes of the heap used by the malloc() and free() device system calls. Setting **CU_LIMIT_MALLOC_HEAP_SIZE** must be performed before launching any kernel that uses the malloc() or free() device system calls, otherwise **CUDA_ERROR_INVALID_VALUE** will be returned. This limit is only applicable to devices of compute capability 2.0 and higher. Attempting to set this limit on devices of compute capability less than 2.0 will result in the error **CUDA_ERROR_UNSUPPORTED_LIMIT** being returned.
- **CU_LIMIT_DEV_RUNTIME_SYNC_DEPTH** controls the maximum nesting depth of a grid at which a thread can safely call cudaDeviceSynchronize(). Setting this limit must be performed before any launch of a kernel that uses the device runtime and calls cudaDeviceSynchronize() above the default sync depth, two levels of grids. Calls to cudaDeviceSynchronize() will fail with error code cudaErrorSyncDepthExceeded if the limitation is violated. This limit can be set smaller than the default or up the maximum launch depth of 24. When setting this limit, keep in mind that additional levels of sync depth require the driver to reserve large amounts of device memory which can no longer be used for user allocations. If these reservations of device memory fail, **cuCtxSetLimit** will return **CUDA_ERROR_OUT_OF_MEMORY**, and the limit can be reset to a lower value. This limit is only applicable to devices of compute capability 3.5 and higher. Attempting to set this limit on devices of compute capability less than 3.5 will result in the error **CUDA_ERROR_UNSUPPORTED_LIMIT** being returned.
- **CU_LIMIT_DEV_RUNTIME_PENDING_LAUNCH_COUNT** controls the maximum number of outstanding device runtime launches that can be made from the current context. A grid is outstanding from the point of launch up until the grid is known to have been completed. Device runtime launches which violate this limitation fail and return cudaErrorLaunchPendingCountExceeded when cudaGetLastError() is called after launch. If more

pending launches than the default (2048 launches) are needed for a module using the device runtime, this limit can be increased. Keep in mind that being able to sustain additional pending launches will require the driver to reserve larger amounts of device memory upfront which can no longer be used for allocations. If these reservations fail, **cuCtxSetLimit** will return **CUDA_ERROR_OUT_OF_MEMORY**, and the limit can be reset to a lower value. This limit is only applicable to devices of compute capability 3.5 and higher. Attempting to set this limit on devices of compute capability less than 3.5 will result in the error **CUDA_ERROR_UNSUPPORTED_LIMIT** being returned.

Parameters:

limit - Limit to set
value - Size of limit

Returns:

CUDA_SUCCESS, **CUDA_ERROR_INVALID_VALUE**, **CUDA_ERROR_UNSUPPORTED_LIMIT**, **CUDA_ERROR_OUT_OF_MEMORY**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSynchronize**

CUresult cuCtxSetSharedMemConfig (CUsharedconfig config)

On devices with configurable shared memory banks, this function will set the context's shared memory bank size which is used for subsequent kernel launches.

Changed the shared memory configuration between launches may insert a device side synchronization point between those launches.

Changing the shared memory bank size will not increase shared memory usage or affect occupancy of kernels, but may have major effects on performance. Larger bank sizes will allow for greater potential bandwidth to shared memory, but will change what kinds of accesses to shared memory will result in bank conflicts.

This function will do nothing on devices with fixed shared memory bank size.

The supported bank configurations are:

- **CU_SHARED_MEM_CONFIG_DEFAULT_BANK_SIZE**: set bank width to the default initial setting (currently, four bytes).
- **CU_SHARED_MEM_CONFIG_FOUR_BYTE_BANK_SIZE**: set shared memory bank width to be natively four bytes.
- **CU_SHARED_MEM_CONFIG_EIGHT_BYTE_BANK_SIZE**: set shared memory bank width to be natively eight bytes.

Parameters:

config - requested shared memory configuration

Returns:

CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**, **CUDA_ERROR_INVALID_VALUE**

Note:

Note that this function may also return error codes from previous, asynchronous launches.

See also:

cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetLimit**, **cuCtxSynchronize**, **cuCtxGetSharedMemConfig**, **cuFuncSetCacheConfig**,

CUresult cuCtxSynchronize (void)

Blocks until the device has completed all preceding requested tasks. **cuCtxSynchronize()** returns an error if one of the preceding tasks failed. If the context was created with the

CU_CTX_SCHED_BLOCKING_SYNC flag, the CPU thread will block until the GPU context has finished its work.

Returns:
CUDA_SUCCESS, **CUDA_ERROR_DEINITIALIZED**, **CUDA_ERROR_NOT_INITIALIZED**, **CUDA_ERROR_INVALID_CONTEXT**

Note:
Note that this function may also return error codes from previous, asynchronous launches.

See also:
cuCtxCreate, **cuCtxDestroy**, **cuCtxGetApiVersion**, **cuCtxGetCacheConfig**, **cuCtxGetDevice**, **cuCtxGetLimit**, **cuCtxPopCurrent**, **cuCtxPushCurrent**, **cuCtxSetCacheConfig**, **cuCtxSetLimit**

Author

Generated automatically by Doxygen from the source code.